(2)

Distributed Decomposition of Block-Angular Linear Programs on a Hypercube Computer*

James K. Ho
S. Kingsley Gnanendran

Management Science Program
College of Business Administration
University of Tennessee
Knoxville, TN 37996-0562

July, 1989

*Research supported in part by the Office of Naval Research under grant N00014-89-J-1528.

Abstract

Algorithms based on the Dantzig-Wolfe decomposition principle for linear programs are implemented on an Intel iPSC-2 Hypercube computer with 64-processors. Computational results with block-angular linear programs from diverse applications are reported. They indicate that the approach of distributed computation on relatively inexpensive multiple processor computers may be very cost-effective for large, structured linear programs. It is also shown that by studying certain characteristics of the interaction among the master and subproblems, one can select algorithms that best exploit the parallel processing environment.

Keywords: Large-Scale Systems; Linear Programming Decomposition; Parallel Processing; Hypercube Computers.

1. Introduction

Many complex systems consisting of independent subsystems coupled by global constraints can be modeled as linear programs with the block-angular structure. The decomposition principle of Dantzig and Wolfe [1] leads to algorithms that transform the original problem into a sequence of subproblems corresponding to the uncoupled subsystems. The subproblems are coordinated by a master problem corresponding to the global constraints through primal (proposals) and dual (prices) information. While it has been obvious that the subproblems can be solved simultaneously for algorithmic efficiency, it is not until recently that advances in computer technology make such an approach realizable.

Advances in VLSI (very large-scale integration) for digital circuit design are leading to much less expensive and much smaller computers. They have also made it possible to build a variety of "supercomputers" consisting of many small computers combined into an array of concurrent processors. We shall refer to such an architecture as multicomputers. Each individual processor is called a node. At this writing, multicomputers with up to 128 nodes are commercially available from at least half a dozen manufacturers. Typically, the nodes are the same kind as those used in high-end microcomputers and are relatively inexpensive. Significant computational power can be obtained by making many of them work in parallel at costs that are much lower than an equivalent single processor. Obviously, the effectiveness of the approach depends on whether an application can be reduced to a well-balanced distribution of asynchronous tasks on the nodes. Linear programming decomposition fits naturally into this framework.

In [6], initial experience of implementing Dantzig-Wolfe decomposition on an experimental multicomputer, the CRYSTAL system [2] at the University of Wisconsin-Madison was reported. On the completion of that phase of our work, commercial multicomputers are already becoming available. These are supported by major vendors and are expected to become prevalent in the foreseeable future. For that reason, we have continued our research on a particularly popular architecture, the hypercube. Since we have much better control over the computing environment than in the earlier experimental system, considerable improvement in the measurement of performance as well as the comparison of results is achieved. In this paper, we describe briefly the hypercube environment and the design of the LP decomposition code DECUBE which runs on Intel's iPSC-2 Hypercube with 64 processors. Computational results on problems from diverse applications are reported. It is also shown that by observing certain patterns in the interaction among the master and subproblems, one can predict the efficiency of algorithmic strategies designed to exploit parallel processing.

2. Hypercube Multicomputers

This architecture is essentially a network of 2^n processors interconnected in a binary n-cube (or hypercube) topology. The connections for $n \leq 4$ are illustrated in Figure 1. Each processor (or node) has its local memory and runs asynchronously of the others. Communication is done by means of messages. A node can communicate directly with its n neighbors. Messages to more distant nodes are routed through intermediate nodes. The hypercube topology provides an efficient balance between the costs of connection and the benefits of direct linkages. Usually, a host computer serves as an administrative console and as a gateway to the hypercube for users.

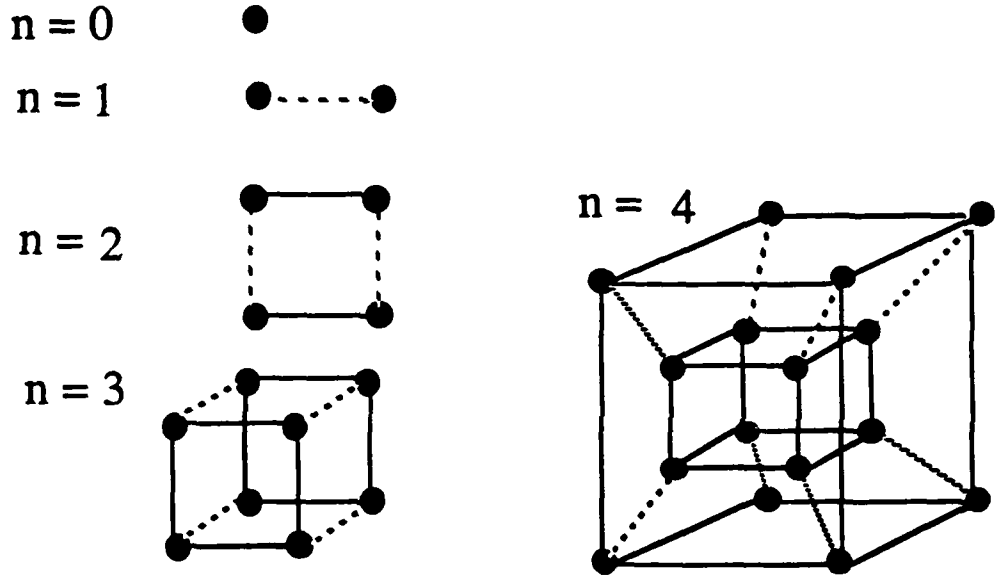

Figure 1. Hypercubes of dimension $n \leq 4$.

For the work reported in this paper, we used an Intel iPSC/2 d6 with 64 nodes at the Oak Ridge National Laboratory. Each node consists of Intel's 32-bit 80386 CPU (4 MIPS) coupled with a 80387 (300 Kflops) numeric coprocessor for floating point acceleration. It has 4 MBytes of local memory. The hypercube (or Cube) is accessed via a host (or System Resource Manager) which is also a 80386-based system with 8 MByte memory and a 140 MByte hard disk. The operating system on the host is the UNIX System V/386 (Release 3.0). The data transfer rate between the System Resource Manager and the Cube has a peak value of 2800 KBytes/sec.

Although the nodes are physically connected as a hypercube, a trade-marked routing network called DIRECT-CONNECT provides essentially uniform communication linkages among all the nodes. The earlier "store and forward" method used in first-generation hypercubes is replaced by a

hardware switching system, the Direct-Connect Module (DCR) on each node. Each DCR provides seven full-duplex channels for internodal communication and one for connection to the System Resource Manager or I/O devices. The network uses a special algorithm for messages longer than 100 bytes. It first sends a header message to the destination node. This header sets gates in each DCR on the intermediate nodes to clear a data path for the message. Once communication with the destination node is established with acknowledgment of receipt of the header, the message is send through at essentially hardware data transfer rates. The implication of this improved technology is that computational efficiency is essentially independent of the problem domain to machine topology mapping. The hypercube can be programmed as an ensemble of processors with an arbitrary communications network in which each node can communicate more or less uniformly with all other nodes.

The host machine allows the user to perform the following tasks.

i) To edit, compile and link host/node programs.
ii) To access and release the cube (or a partition thereof).
iii) To execute the host program.
iv) To start or kill processes on the cube.

Operations peculiar to the hypercube are controlled either by UNIX-type commands (iPSC/2 commands) or by extensions to standard programming languages such as Fortran and C (iPSC/2 routines). The iPSC/2 commands are used to gain access to the cube, to load, start or kill cube processes and to relinquish access to the cube. These commands may be input from a terminal or they may be invoked using a shell script. The iPSC/2 routines, on the other hand, are mainly used to manage internodal messages. Nevertheless, it should be noted that almost all of the tasks that can be performed with iPSC/2 commands can also be accomplished from within the user programs by iPSC/2 routines with similar names. The iPSC/2 commands and routines for the Fortran programming environment are documented in [12].

To execute a typical parallel program, the following steps are used.

I) Compile and link the host and node programs to create executable modules.
II) Obtain a partition of the cube (a subcube) of suitable size by invoking the GETCUBE command. The user has the option of providing a name to identify this partition. For example, the command

```
% getcube -c sugar -t d3
```

allocates to the user an exclusive subcube named sugar with dimension 3 (i.e. 8 processors) identified by the node numbers 0, 1, ..., 7.

III) Run the host program by invoking the name of the executable host module. Node programs are loaded on to the appropriate nodes at runtime in response to calls to the LOAD subroutine in the host program.

IV) On termination, kill all node processess and flush messages by invoking the KILLCUBE command.

V) Relinquish access to the subcube by the RELCUBE command.

Internodal and host-to-node communication is done by subroutine calls in the corresponding programs. The subroutine to send messages is called CSEND. Its arguments are:

- message type (ID)
- message location (address)
- message length in bytes
- destination node ID
- destination process ID.

The subroutine to receive messages is called CRECV. Its arguments are:

- message type (ID)
- address of buffer for storing message
- length of buffer in bytes.

Both CSEND and CRECV are blocking commands in the sense that the calling process halts until the message has been transmitted and received, respectively. Non-blocking versions of these commands are also provided as ISEND and IRECV respectively.

Other features necessary for our purpose are the following functions:

IPROBE( ) - indicating whether a message of a particular type has been received;

MYHOST( ) - indicating the node ID of the host;

MCLOCK( ) - returning elapsed times on the nodes and CPU times on the host; and

MSGWAIT( ) - blocking the calling process until the outgoing message has been copied to the operating system buffer.

3. DECUBE: an LP Decomposition Code on the Hypercube

As the building blocks of our parallel implementation, we use DECOMP: a Fortran code of the Dantzig-Wolfe decomposition algorithm. This code was first assembled in 1973 by Carlos Winkler at the Systems Optimization Laboratory at Stanford University. It was based on John Tomlin's LPM1 code of the revised simplex method. Since then, DECOMP has been extended and improved over a number of years by James Ho and Etienne Loute at the Center for Operations Research and Econometrics in Belgium. It was used extensively in empirical studies (see e.g. [8]) and as prototype for more advanced, commercial software based implementations (e.g.

DECOMPSX in [7]). A comprehensive documentation of DECOMP is given in [10].

The Dantzig-Wolfe decomposition algorithm (see e.g. [1], [4], [7]]) is assumed well known and its description is omitted here. While many refined features have been used to make DECOMP computationally efficient and robust, we list below only the major ones. Since all of these techniques are also used in the advanced implementation DECOMPSX described in [7], the reader may refer to that paper for further details.

i) Data is disk resident. The LP to be solved (either the master problem or a subproblem) is read into memory. Subsequent modifications are written to disk.

ii) All matrices are stored in sparse form using column packing. The coupling coefficients for the variables of a subproblem are included in that subproblem as nonbinding constraints whose updated right-hand-side gives a proposal to the master problem.

iii) The prices π are incorporated directly into the dual variables of the subproblems without actual modification of its objective function.

The basic idea behind the design of DECUBE is to process the master and subproblems asynchronously on the nodes of the hypercube. In the current version, each subproblem is assigned to one node and the master problem is also handled by a dedicated node. To allow for situations where there are more subproblems than nodes, later versions will have built-in mechanisms to distribute the subproblems appropriately. All relevant data for a subproblem will be core resident on its node. Coordinating information between the master and subproblems in the form of prices and proposals is communicated as internodal messages. Although DECUBE contains several environment-specific statements (such as communication primitives), its modular design makes adaptation to other distributed-memory machines relatively straightforward.

DECUBE consists of three Green Hills Fortran ([12], [13]) programs: *Host, Master* and *Sub*. The *host* program runs on the host machine and is the process that controls the entire parallel program. The *master* program solves the Dantzig-Wolfe restricted master problem and runs on node 0. The *sub* program solves a Dantzig-Wolfe subproblem and runs on nodes 1, 2, ..., LMAX where LMAX is the number of blocks in the block-angular linear program. The master program and LMAX copies of the *sub* program are loaded onto the appropriate nodes at runtime in response to calls to the LOAD subroutine in the *host* program.

In the following tables, we describe the various modules which make up the *host*, *master* and *sub* programs.

Type	Name	Function
Communication	COMM	Open I/O files and set process ID; load *master* and *sub* programs to appropriate nodes.
	SNDDAT	Send an MPS data section to a specified node.
	RCVTRM	Receive information on problem status from master node on termination of decomposition algorithm; receive timing information from all nodes.
	WRAPUP	Close I/O files; kill all cube processes.
Input	INDATA	Driver for reading LP data from a DECOMP formatted file [10].
	INPUT	Read an MPS data section.
Output	STATS	Print problem status; compute and print performance measures.
	GRAPH	Create a graphics interface file for proposal dispersion charts.

Table 1. Subroutines of the *host* program.

Type	Name	Function
Communication	COMM	Open output file; set process ID.
	RCVDAT	Receive relevant MPS data section from host.
	RCVPRP	Receive proposals from subproblem nodes.
	SNDPRI	Send prices to subproblem nodes.
	SNDRHS	Send Phase 3 right-hand side allocations to subproblem nodes.
	SNDTRM	Receive proposal generation statistics from subproblem nodes; send timing data and problem status to host.
Decomposition	INIT	Initialize arrays, counters.
	INIMAS	Setup initial master problem.
	SLVMAS	Driver for the prices generation process (receive incoming proposals, check optimality, activate simplex algorithm to solve the restricted master problem, and send prices.)
	PACK	Incorporate new proposals into master problem; purge "obsolete" proposals periodically.
Revised Simplex	BTRAN	Backward transformation for prices.
	CHSOL	Check accuracy of solution.
	CHUZR	Ratio tests for column to leave basis.
	FORMC	Check feasibility.
	FTRAN	Forward transformation for updated column.
	INVERT	Refactorization of the basis.
	ITEROP	Print information on simplex iteration.
	NORMAL	Driver for Primal Revised Simplex method.
	PRICE	Pricing for column to enter basis.
	UNPACK	Unpack a column in the LP matrix.
	UPBETA	Update solution.
	WRETA	Update basis factorization.
Phase 3	RESULT	Driver for reconstruction of primal solution to LP.
	UNRAVL	Output solution.

Table 2. Subroutines of the *master* program.

Type	Name	Function
Communication	COMM	Open output file; set process ID.
	RCVDAT	Receive relevant MPS data section from host.
	RCVPRI	Receive prices from master node.
	SNDPRP	Send proposals to master node.
	RCVRHS	Receive Phase 3 right-hand side allocation from master node.
	SNDTIM	Send timing data to host.
	SNDGEN	Send proposal generation statistics to master node.
Decomposition	INIT	Initialize arrays, counters.
	COPY	Copy initial proposal to proposal buffer.
	SLVSUB	Driver to control proposal generation process (receive prices, solve subproblem, and send proposals to master node).
	CHECK	Candidacy test for proposal generation.
	POLICY	Set parameters for proposal generation process according to user-specified strategy.
Revised Simplex	(Same procedures as in the *master* program)	
Phase 3	RESULT	Driver for reconstruction of primal solution to subproblem.
	UNRAVL	Print solution of subproblem.

Table 3. Subroutines of the *sub* program

All statements peculiar to the hypercube have been isolated into separate subroutines. These are the subroutines of the communication type in the above Tables. To adapt DECUBE to a different distributed-memory environment, the user needs only to replace the primitives in these subroutines (such as CRECV, CSEND, IPROBE, LOAD, MYHOST, SETPID, KILLCUBE) with equivalents if available. If exact equivalents are not available, these same subroutines calls may be used to invoke user-written subroutines built upon more basic primitives in the new environment.

The following is a pseudo-code description of DECUBE in what will be referred to as the standard version. Variations of this scheme will be implemented to accomodate different strategies made possible by parallel computation.

DECUBE: *Host* Program

Step 1: Data Input

1.1 Read master problem data and send to node 0.

1.2 For i = 1 to LMAX,

1.2.1 read data of subproblem i;

1.2.2 send data to node i;

Step 2: Output

2.1 Receive problem status and timing data from node 0.

2.2 Receive timing data from nodes 1 to LMAX.

2.3 Compute performance measures.

2.4 Print problem status and performance measures.

2.5 Stop.

DECUBE: *Master* Program

Step 1: Initialization

1.1 Wait to receive master problem data from host.

1.2 Wait to receive initial proposals from subproblem nodes.

1.3 Incorporate initial proposals into master problem.

Step 2: Iterations

2.1 Phase 1

2.1.1 Set Phase 1 objective.

2.1.2 CYCLE (see below) until termination.

2.1.3 If infeasible, stop.

2.2 Phase 2

2.2.1 Set Phase 2 objective.

2.2.2 CYCLE until termination.

2.2.3 If unbounded, stop.

Step 3: Phase 3

3.1 Compute allocations for subproblems.

3.2 Send allocations to suproblem nodes.

3.3 Output master problem solution.

3.4 Stop.

CYCLE:

C.1 Set flag to 'continue'.

C.2 Solve master problem. Exit CYCLE if unbounded.

C.3 If [primal-dual gap < tolerance] or [no more proposals], set flag to 'done'.

C.4 Send price buffer and flag to all subproblem nodes.

C.5 Exit CYCLE if flag = 'done'.

C.6 Wait to receive proposals from all subproblem nodes.

C.7 Incorporate proposals into master problem.

C.8 Return to C.2.

DECUBE: *Sub* Program

Step 1: Initialization

1.1 Wait to receive subproblem data from host.

1.2 Solve subproblem.

1.3 If infeasible, stop;
else generate proposal.

1.4 Send proposal to master node.

Step 2: Iteration

2.1 Wait for prices and flag from master node.

2.2 If flag = 'done' go to Step 3; else solve subproblem.

2.3 Generate proposals, if any.

2.4 Send proposals to master node.

2.5 Return to 2.1.

Step 3: Phase 3

3.1 Receive allocation from master node.

3.2 Compute subproblem solution.

3.3 Output subproblem solution.

3.4 Stop.

Apart from specific adaptation to the hypercube environment, the design of DECUBE is very similar to that of DECOMPAR [6] which ran on the experimental multicomputer known as CRYSTAL [2] until the latter system became obsolete and went out of commission. While DECOMPAR provided valuable initial results, DECUBE benefits from a much better controlled computing environment and allows very accurate measures of performance of new techniques in LP decomposition using parallel processing. Currently, DECUBE is dimensioned for block-angular LP's with up to 30 subproblems, each with up to 1200 rows, 4000 columns and 200,000 nonzeros for coefficient matrix and basis data. The master problem can have up to 200 rows besides the objective and convexity rows. In terms of total problem size, DECUBE can handle up to 30,000 rows, 120,000 columns and 3.6 million non-zeros in the coefficient matrix. These dimensions can be further extended within the system configuration described in § 2.

4. Performance Measures

The primary purpose of parallel processing is to speed up computing time relative to conventional sequential computation. Assuming that a number of processors (p) are available and allocated to the problem on hand, we seek to compare the parallel time utilizing multiple processors with the sequential time on one processor. The parallel time is the duration from start to finish of the solution process. In terms of CPU times, we are therefore interested in the *disjoint union* (i.e nonoverlapping total) of CPU times on all the processors. This is denoted by TPAR. The sum of all CPU times is used as a lower bound on the sequential time. This is the least amount of time required to solve the same problem by one processor (assuming that a single processor can indeed handle all the tasks without additional resources). It is denoted by TSEQ. The *speedup* (S) in using p processors instead of one is given by TSEQ/TPAR. The *efficiency* (E) is given by S/p.

Actual sequential time is likely to be much higher than TSEQ. For example, in linear programming decomposition, the entire problem is usually too large to fit in the local memory of a single processor. The data will have to be disk resident (as with DECOMP described above) and the appropriate portion fetched into memory as required. This will naturally increase the total solution time significantly. Therefore by using TSEQ as a basis of comparison, the speedup measures that we obtain for DECUBE are conservative and tend to underestimate actual performance improvements.

In [3], [5] and [14], various information schemes to control the coordinating process of LP decomposition were analyzed and tested using DECOMPAR. Based on those earlier results, we have implemented two schemes in DECUBE. The first is known as the Basic Information Scheme (BIS) and is a straightforward parallelization of the Dantzig-Wolfe decomposition algorithm. The master problem determines prices (π) on the coupling constraints while the subproblems are idle. With these prices, the subproblems are processed concurrently to generate proposals while the master problem is idle. An illustration of this information scheme with three subproblems is shown in Figure 2. In this and the following figure, the black rectangles are used to denote the busy time of the master problem, and the shaded rectangles the busy time of the subproblems. The white areas denote idle time. By nature of the decomposition algorithm with the basic information scheme, the theoretical limit of E is not 1. This is because a cycle in the algorithm consists of first a master problem and then the subproblems. Only the latter are processed in parallel in BIS. Suppose a problem with r blocks takes t cycles in which the average time for the master is m and the average time for a subproblem is s. Then the total sequential time is approximately $t (m + rs)$. The total parallel time using one processor for each of the master and subproblems is approximately $t (m+s)$. Therefore, letting $p = r + 1$, we have

$$E \approx (m + r s) / (m + s)(r + 1).$$

For m close to s, $E \approx 1/2$.

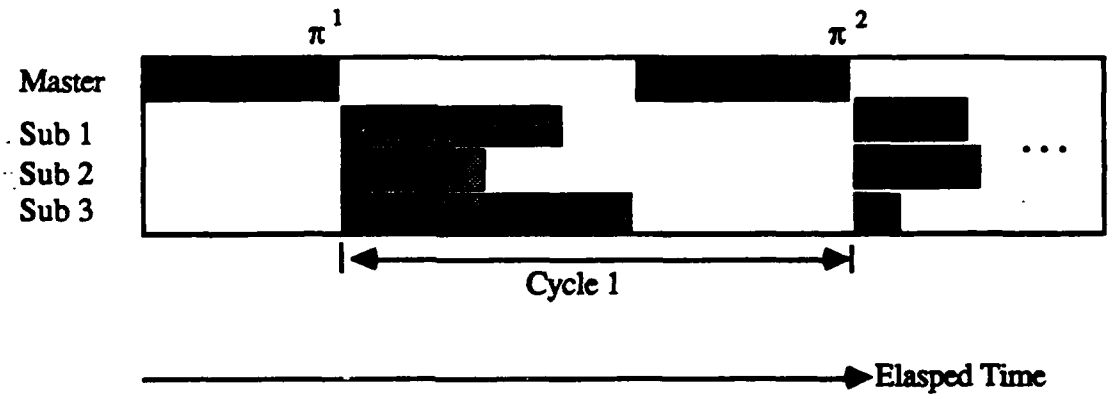

Figure 2. Basic Information Scheme (BIS)

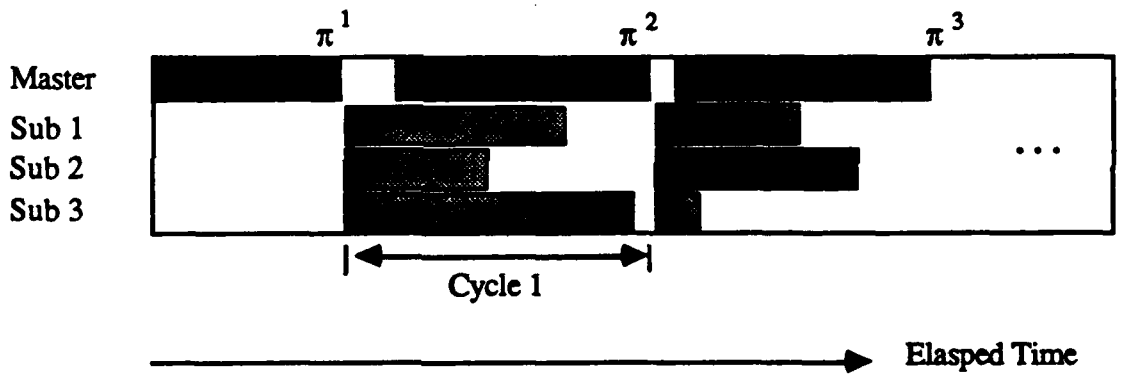

Figure 3. Early Start Information Scheme (ESIS)

In general, many proposals may be generated by a subproblem during a cycle. Their arrivals are dispersed over the busy period of the subproblem. Therefore, instead of letting the master problem remain idle until all the subproblems are done, we can have it started as soon as the first proposal becomes available. Note that eventually, the master would have received the same set of proposals for a particular cycle as in BIS. The only difference is that it gets a head start. We call this the Early Start Information Scheme (ESIS). The early processing of proposals to determine new prices does not guarantee solving the master problem faster. It may even require more time than starting the process after *all* the proposals are received. However, it is hoped that the overall *cycle time* would be shorter due to increased concurrency between the master and subproblems. Note that

the proposals and prices generated in ESIS are identical to those in BIS. Therefore, any effect that ESIS may have on the elapsed time of the solution process is not due to the information but the early processing of the proposals. An obvious case in which ESIS is superior is illustrated in Figure 3. Comparing Figures 2 and 3, we see that although the time required to generate π^2 in ESIS is longer than that in BIS, the overall time for cycle 1 is shorter. It is of course conceivable that the early start might lead to a much longer solution path to π^2 so that the cycle time turns out to be longer than that in BIS.

Other information schemes involve sending intermediate prices from the master problem to the subproblems and have the latter generate new proposals according to the more frequently updated prices. However, initial empirical results showed that any benefit from such improved feedback of information tend to be offset by the additional volume of information to be processed. As a result, such intermediate prices information schemes rarely outperform BIS or ESIS. For this reason, we have included only BIS and ESIS in the current version of DECUBE.

Let the parallel time with BIS be denoted by TPAR(BIS). From this we can obtain an upper bound on the potential improvement with ESIS. Since the best we can hope to do is to have all the master problem times overlap with the subproblem busy periods, TPAR(BIS) minus all the master problem times gives a lower bound on TPAR(ESIS), the actual parallel time using ESIS. It turns out we can predict to a certain extent how well ESIS will do by studying the timing pattern of the problem as represented in Figure 2. This will be demonstrated in the next section where we compare TSEQ, TPAR(BIS) and TPAR(ESIS) for a diverse collection of block-angular linear programming models.

5. Computational Results

The experiments were organized around three sets of test problems for the following purposes:

I) to validate DECUBE and observe speedup and efficiency measures for a collection of small to medium size test problems from diverse applications;

II) to apply knowledge from (I) to predict the performance on new applications; in particular, two planning models in electric power generation [3]; and

III) to test DECUBE on large problems from a material requirements planning model [9] and to compare with conventional software.

For experiment (I) ten small to medium size test problems comprising what we call the standard set are used. Their characteristics are listed in Table 4.

Problem	Blocks		Rows		Columns	% Density
	Actual	Natural	Coupling	Total		
DEEP1	6	6	16	101	265	19.0
DEEP2	4	4	11	101	226	18.9
MEAT12	6	12	46	381	692	1.3
MEAT31	8	31	11	384	961	1.3
MEAT43	9	43	16	648	1253	0.7
FORESTRY	6	6	11	402	1005	1.0
SCORPION	6	6	45	389	747	0.7
DYNAMICO	5	10	10	678	1176	0.7
MRP3	3	3	31	301	823	0.9
MRP5	5	5	61	961	2701	0.4

Table 4. Characteristics of the Standard Test Problems

DEEP1 and DEEP2 are randomly generated problems with dense blocks. All the others are from real applications. The MEATxx examples are for multiproduct ingredient mix optimization in the meat processing industry. FORESTRY is from a forest management model. SCORPION is from a French energy model. DYNAMICO is a model of world trade and development from the United Nations. MRP3B and MRP5B are material requirements planning problems in production management. The column counts in Table 4 include one logical column per row. For some problems, several natural blocks are grouped together to reduce the actual number of subproblems required. Tables 5 and 6 record the performance of DECUBE on the standard problems with BIS and ESIS respectively. The average efficiency is 36.6% for BIS and 38.5% for ESIS. The last column in Table 6 gives the ratios of these efficiencies as a measure of improvement of ESIS over BIS. Note that while the average improvement is only about 5%, it will be of practical interest to identify both favorable and unfavorable applications for ESIS. To a certain extent, this can be achieved by studying the proposals generation pattern of the subproblems and the relative complexities of the master and subproblems. The timing statistics from DECUBE with BIS are used to plot the time at which each proposal is generated as well as the duration of each master problem. This is called a BIS chart of the problem. Figure 4 shows the chart of a favorable case: DYNAMICO. The master and subproblem times are comparable and the proposals are evenly dispersed. These are indications that a significant portion of the master times can be made concurrent with subproblem busy times. An unfavorable case, MEAT12 is shown in Figure 5. Here the master times dominate and starting early does not help. For similar reasons, the reverse situation of trivial master times is also unfavorable for ESIS. Note that the BIS charts can also be constructed with timing data from a sequential implementation of decomposition such as DECOMP.

PROBLEM	p	TSEQ	TPAR(BIS)	SPEED-UP	EFFICIENCY
		(CPU seconds)		TSEQ/TPAR	(%)
DEEP1	7	31.87	18.78	1.70	24.24
DEEP2	5	21.27	11.66	1.82	36.46
MEAT12	7	64.74	45.98	1.41	20.11
MEAT31	9	14.20	4.45	3.19	35.41
MEAT43	10	21.55	5.36	4.02	40.23
FORESTRY	7	84.13	29.55	2.85	40.68
SCORPION	7	8.30	2.63	3.15	45.03
DYNAMICO	6	116.20	39.44	2.95	49.10
MRP3	4	18.54	11.24	1.65	41.23
MRP5	6	193.35	96.23	2.01	33.49
				Average:	36.60

Table 5. Performance of DECUBE(BIS) on Standard Problems

PROBLEM	p	TSEQ	TPAR(ESIS)	EFFICIENCY	ESIS/BIS
		(CPU seconds)		(%)	
DEEP1	7	31.87	22.07	20.63	0.85
DEEP2	5	21.27	9.40	45.25	1.24
MEAT12	7	64.74	41.71	22.17	1.10
MEAT31	9	14.20	4.31	36.58	1.03
MEAT43	10	21.55	5.07	42.52	1.06
FORESTRY	7	84.13	29.44	40.82	1.00
SCORPION	7	8.30	2.59	45.89	1.02
DYNAMICO	6	116.20	34.53	56.08	1.14
MRP3	4	18.54	11.11	41.72	1.01
MRP5	6	193.35	96.85	33.27	0.99
			Average:	38.49	1.05

Table 6. Performance of DECUBE(ESIS) on Standard Problems

Experiment (II) is an example of applying BIS chart analysis to new problems to predict the performance of DECUBE. Two models in electric power system planning [3] are used. The first is a problem in the economic dispatch of electric power generators to satisfy system demands and constraints imposed by, for example, regulation and reserve margin requirements. The number of blocks is equal to the number of generators to be dispatched. The characteristics of five test cases are given in Table 7. In each case, the natural blocks in the LP are grouped into ten subproblems for convenience.

Problem	Blocks		Rows		Columns	% Density
	Actual	Natural	Coupling	Total		
EGD017	10	17	8	295	431	0.57
EGD034	10	34	8	582	854	0.29
EGD051	10	51	8	869	1277	0.20
EGD085	10	85	8	1443	2123	0.12
EGD170	10	170	8	2878	4238	0.06

Table 7. Characteristics of the Electric Generation Dispatch Problems

Figure 6 shows the BIS chart of problem EGD051. In terms of the dispersion of proposals and the complexity of the master problem relative to those of the subproblems, this picture is typical of the five cases. In all of them, we observe that the master takes roughly two-thirds of the longest subproblem time in the same cycle; that the proposals are widely spread; and that the subproblem loads are rather uneven. By starting the master problems early and incorporating proposals as they are being generated, we expect ESIS to perform well on these problems.

PROBLEM	p	TSEQ	TPAR(BIS)	SPEED-UP	EFFICIENCY
		(CPU seconds)		TSEQ/TPAR	(%)
EGD017	11	3.23	1.16	2.78	25.27
EGD034	11	9.67	2.85	3.40	30.89
EGD051	11	17.08	4.08	4.19	38.09
EGD085	11	41.76	7.68	5.43	49.41
EGD170	11	199.55	28.89	6.91	62.79
				Average:	41.29

Table 8. Performance of DECUBE(BIS) on Electric Generation Dispatch Problems.

PROBLEM	p	TSEQ	TPAR(ESIS)	EFFICIENCY	ESIS/BIS
		(CPU seconds)		(%)	
EGD017	11	3.23	1.23	23.82	0.94
EGD034	11	9.67	2.72	32.34	1.05
EGD051	11	17.08	3.73	41.66	1.09
EGD085	11	41.76	7.07	53.68	1.09
EGD170	11	199.55	29.15	62.23	0.99
			Average:	42.75	1.03

Table 9. Performance of DECUBE(ESIS) on Electric Generation Dispatch Problems.

Observe that the efficiency of parallel decomposition increases with the problem size in this case. This is very promising for truly large-scale problems. It should be remarked that in general, it is difficult to obtain such correlations. Our test problems from the electric generation dispatch model illustrate this fact nicely because they are well controlled progressions in terms of dimensions and complexity. Also, as suggested by the BIS chart, ESIS performs better than BIS (43% compared to 41% on the average).

The second electric power generation model is one in multiregion capacity expanasion planning. Each block in the LP represents a region which has to select capacity additions that minimize the present value of investment and operation costs subject to reserve margin, customer load, capacity mix, land availability, fuel availability and emission constraints. The regions can trade electric power among themselves. The coupling constraints state that net import and export should be balanced. Again, five test cases are used. Their characteristics are given in Table 10.

Problem	Blocks	Coupling Rows	Total Rows	Columns	%Density
MEGE02	2	15	349	614	1.29
MEGE04	4	15	679	1105	0.58
MEGE06	6	15	1011	1698	0.41
MEGE08	8	15	1343	2251	0.31
MEGE10	10	15	1675	2847	0.25

Table 10. Characteristics of the Electric Generation Expansion Problems

Figure 5 shows the BIS chart for MEGE10. Note that before the first master is started (time zero in the figure), each subproblem is solved to submit a proposal. For importing regions, this first proposal contains all the information on the deficit in the region. In subsequent cycles, only potential exporting regions will have new proposals describing what they can offer in response to coordinating prices. In practice of course, regions may not have to be pure importers or exporters over time. Our test problems are set up this way for simplicity of data generation. However, if this assumption does hold, it can be used to great computational advantage by essentially releasing the nodes corresponding to the importing regions.

Observe that the master times are trivial compared to the subproblem times in this model. That means there is not much to gain at all by starting the master problems early. Therefore, we do not expect ESIS to perform better than BIS on these problems. The results, as reported in Tables 11 and 12 respectively, show that indeed, the two information schemes are almost identical in this case. Note also that while the problems progress in size, they represent diverse scenarios of various complexities. Therefore, there is no obvious correlation between size and efficiency here.

PROBLEM	p	TSEQ	TPAR(BIS)	SPEED-UP	EFFICIENCY
		(CPU seconds)		TSEQ/TPAR	(%)
MEGE02	3	8.45	7.26	1.16	38.81
MEGE04	5	13.13	7.74	1.70	33.96
MEGE06	7	17.24	5.81	2.96	42.36
MEGE08	9	24.83	7.11	3.49	38.82
MEGE10	11	33.29	7.46	4.46	40.55
				Average:	38.90

Table 11. Performance of DECUBE(BIS) on Electric Power Expansion Problems.

PROBLEM	p	TSEQ	TPAR(ESIS)	EFFICIENCY	ESIS/BIS
		(CPU seconds)		(%)	
MEGE02	3	8.45	7.31	38.55	0.99
MEGE04	5	13.13	7.76	33.87	1.00
MEGE06	7	17.24	5.85	42.12	0.99
MEGE08	9	24.83	7.10	38.88	1.00
MEGE10	11	33.29	7.35	41.15	1.01
			Average:	38.91	1.00

Table 12. Performance of DECUBE(ESIS) on Electric Power Expansion Problems.

Experiment (III) attempts to use DECUBE on larger problems and compare to some extent its performance with that of an commercial LP code. We consider the class of production and inventory optimization problems arising from material requirements planning. Typically, they involve many end-products, each one comprising of many component parts. The parts, in turn, are made up of other components, and so forth. The manufacturing or assembly of any component may require certain limited resources such as machine capacity, labor hours, etc. Given the exogeneous demands for the end-products as well as components (e.g. as spare parts) over a finite planning horizon, the problem is to determine the minimum cost production and inventory schedule subject to resource availability constraints. The block-angular LP model is well-known (see e.g. [9]). However, the dimensions are found to be prohibitive in practice. For instance, the plannning of an operation over 10 time periods with 30 end-products, each with an average of 100 parts, gives rise to 30,000 constraints besides the coupling capacity constraints. While such problems can be solved routinely on mainframe computers, the preferred environment for operations management is typically in the range of mini- and supermini-computers. We have tested DECUBE on the iPSC/2 hypercube with MRP problems having up to the above stated dimensions. Prelimary comparison with the performance of MPSX/370 on an IBM 3090-200E with vector support demonstrates that distributed decomposition is indeed going to be a viable and cost-effective approach.

Five pairs of problems are used in the experiment. In each case, the first problem represents 9 capacities over 10 periods for a total of 90 coupling constraints. The second problem represents 19 capacities over 10 periods for a total of 190 coupling constraints. Each all cases, each block has 1000 rows. The characteristics of the MRP test problems are summarized in Table 13.

Problem	Blocks	Coupling Rows	Total Rows	Columns	%Density
MRP05A	5	91	5091	14591	0.099
MRP05B	5	191	5191	14691	0.156
MRP10A	10	91	10091	29091	0.050
MRP10B	10	191	10191	29191	0.080
MRP15A	15	91	15091	43591	0.034
MRP15B	15	191	15191	43691	0.054
MRP20A	20	91	20091	58091	0.025
MRP20B	20	191	20191	58191	0.054
MRP30A	30	91	30091	87091	0.017
MRP30B	30	191	30191	87191	0.027

Table 13. Characteristics of the Material Requirements Planning Problems

PROBLEM	p	TSEQ (CPU seconds)	TPAR(BIS)	SPEED-UP TSEQ/TPAR	EFFICIENCY (%)
MRP05A	6	2109.64	467.16	4.52	75.26
MRP05B	6	2484.91	527.62	4.71	78.49
MRP10A	11	4104.56	517.05	7.94	72.17
MRP10B	11	4972.55	566.18	8.78	79.84
MRP15A	16	6075.21	531.30	11.43	71.47
MRP15B	16	7141.52	574.02	12.44	77.76
MRP20A	21	8141.63	586.34	13.89	66.12
MRP20B	21	10554.27	869.14	12.14	57.83
MRP30A	31	11864.22	561.85	21.12	68.12
MRP30B	31	15555.75	780.25	19.94	64.31
				Average:	71.14

Table 14. Performance of DECUBE(BIS) on MRP Problems

From Table 14, we observe that DECUBE attained very impressive efficiency (an average of 71% over the ten cases) with this class of problems. As expected, an increase in coupling constraints makes the problem more difficult. However, for both the A- and B-series, the inferred sequential solution time (TSER) grows only (approximately) linearly with the problem size (in terms of total rows). The actual parallel times (TPAR) show that most of the effect of even the linear growth is offset by the efficiency of distributed computation. ESIS turns out to be inferior to BIS for most cases in this set and is omitted.

In order to compare this performance with a commercial LP code on a mainframe supercomputer, we use IBM's MPSX/370 version 2 [11] on an IBM 3090-200E. The vector capability of the machine is exploited by the LP code. The results are listed in Table 15. Note that the solution time grows approximately quadratically in the problem size for MPSX/370. On the largest problems tested, DECUBE is four to five times faster. Also, there is a limit of 32768 constraints for MPSX/370 version 2. Therefore our experiment is already approaching its capacity. On the other hand, DECUBE can be further extended to handle even larger problems. Given the fact that the mainframe costs ten to twenty times as much as the hypercube, our results clearly demonstrate viability of the distributed decomposition approach.

The timing results reported do not include problem generation and data input. For large problems such times are substantial and may actually become the dominant factor in the total turn-around time. For example, using the host machine on the hypercube, it takes approximately 4

minutes to generate and down load each subproblem in MRP30A. With 30 subproblems, this data processing phase takes 2 hours compared to the solution time of under 10 minutes. However, the subproblems can be generated directly on the node machines in parallel reducing the data processing time to 4 minutes.

PROBLEM	Code on DECUBE Intel iPSC/2-d6 (CPU minutes)	MPSX/370 V2 IBM 3090-200E (CPU minutes)
MRP05A	7.79	1.92
MRP10A	8.62	6.43
MRP15A	8.85	14.45
MRP20A	9.77	25.45
MRP30A	9.36	55.44
MRP05B	8.79	1.84
MRP10B	9.44	7.43
MRP15B	9.57	16.96
MRP20B	14.49	31.82
MRP30B	13.00	51.10

Table 15. Comparison of DECUBE and MPSX/370

6. Discussion

The decomposition of block-angular linear programs is a natural candidate for distributed computation using the lastest development in multicomputers. We have shown very promising initial results with DECUBE, an implementation on an Intel iPSC/2 hypercube computer with 64 processors. The prototype software can be used to further study the dynamics of information in decomposition algorithms. It also serves as a demonstration of the potential cost-effectiveness of multicomputers for important classes of applications such as material requirements planning. We have shown that in the latter case, DECUBE can significantly outperform conventional software on relatively expensive supercomputers.

Acknowledgement

The authors are indebted to Tom Dunigan and Michael Heath of Oak Ridge National Laboratory for providing access to the iPSC/2 system; and Don Broach for consultation on the IBM 3090 system. Discussions with Tak Lee and R.P. Sundarraj have been very helpful. Part of this work was conducted while the first author was visiting scholar in the Department of Operations Research at Stanford University during the Winter and Spring quarters of 1988. He wishes to thank George Dantzig and Robert Entriken for stimulating discussions.

References

[1] G.B. Dantzig and P. Wolfe, "The decomposition principle for linear programs", *Operations Research* 8 (1960) 101-111.

[2] D. Dewitt, R. Finkel and M. Solomon, "The CRYSTAL multicomputer: design and implementation experience", *IEEE Transactions on Software Engineering* 8 (1987) 953-966.

[3] Electric Power Research Institute, "Decomposition of linear programs using concurrent processing on multicomputers", RP1999-11 Final Report, April, 1989.

[4] J.K. Ho, "Recent advances in the decomposition approach to linear programming", *Mathematical Programming Study* 31 (1987) 119-128.

[5] J.K. Ho and T.C. Lee, "Dynamics of information in distributed decision systems", March 1989 (submitted to *Management Science*).

[6] J.K. Ho, T.C. Lee and R.P. Sundarraj, "Decomposition of linear programs using parallel computation", *Mathematical Programming* 42 (1988) 391-405.

[7] J.K. Ho and E. Loute, "An advanced implementation of the Dantzig-Wolfe decomposition algorithm for linear programming", *Mathematical Programming* 20 (1981) 303-326.

[8] J.K. Ho and E. Loute, "Computational aspects of DYNAMICO: a model of trade and development in the world economy", *Revue Française d'Automatique, Informatique et Recherche Opérationelle* 18 (1984) 403-414.

[9] J.K. Ho and W.A. McKenney, "Triangularity of the basis in linear programs for material requirements planning", *Operations Research Letters* 7 (1988) 273-278.

[10] J.K. Ho and R.P. Sundarraj, *DECOMP: an Implementation of Dantzig-Wolfe Decomposition for Linear Programming*, (to be published by Springer-Verlag, New York.)

[11] IBM Corporation, MPSX/370 Version 2 Program Reference Manual, SH19-6553, 1988.

[12] Intel Corporation, iPSC/2 Fortran Programmer's Reference Manual, Order No. 311019-003, August, 1988.

[13] Intel Corporation, iPSC/2 Green Hills Fortran Language Reference Manual (Preliminary), Order No. 311020-003, August, 1988.

[14] T.C. Lee, Distributed Optimization of Linear Programs using Dantzig-Wolfe Decomposition, Ph.D. Dissertation, University of Tennessee, 1988.

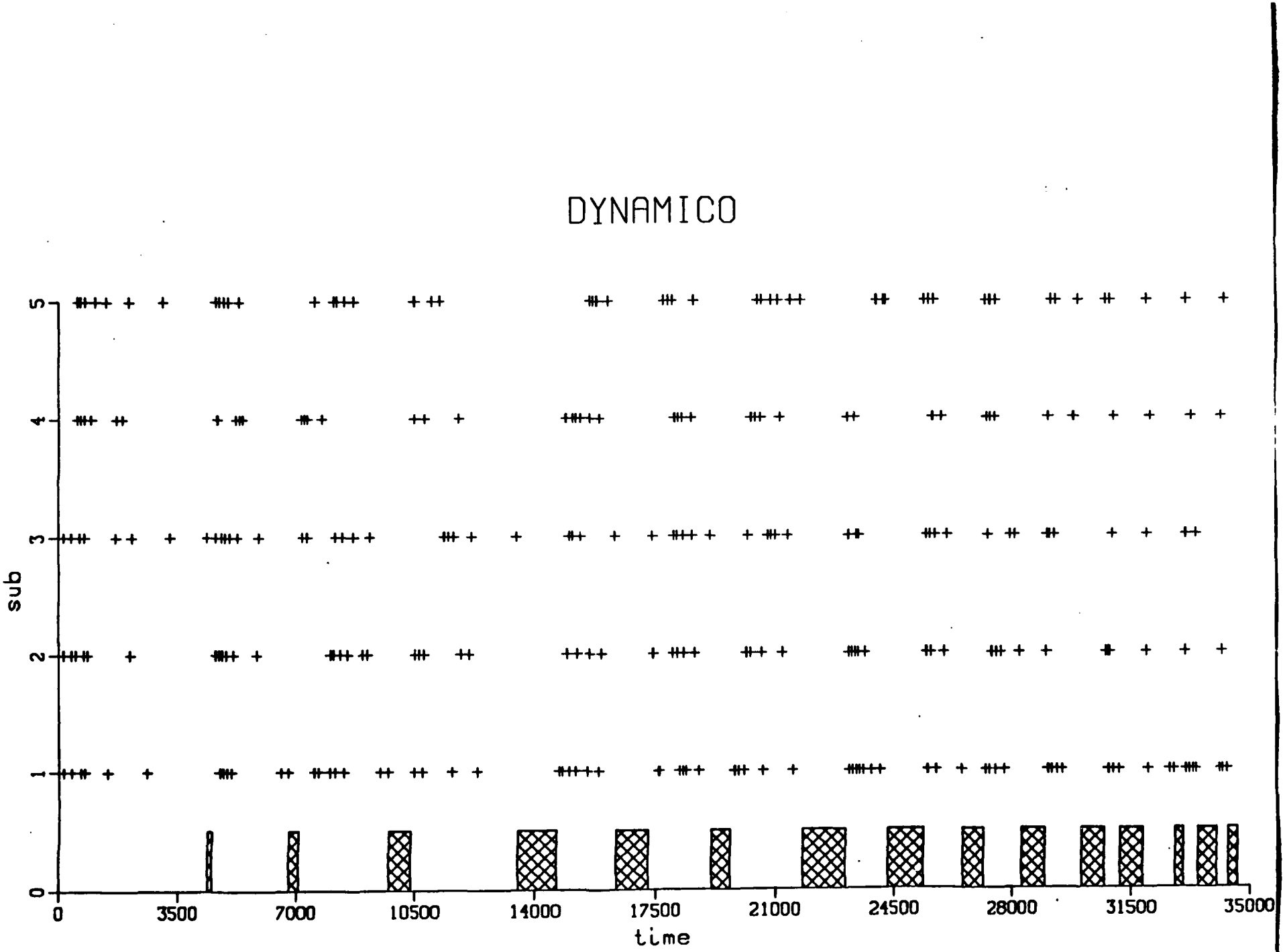

Figure 4 BIS Chart of Problem DYNAMICO

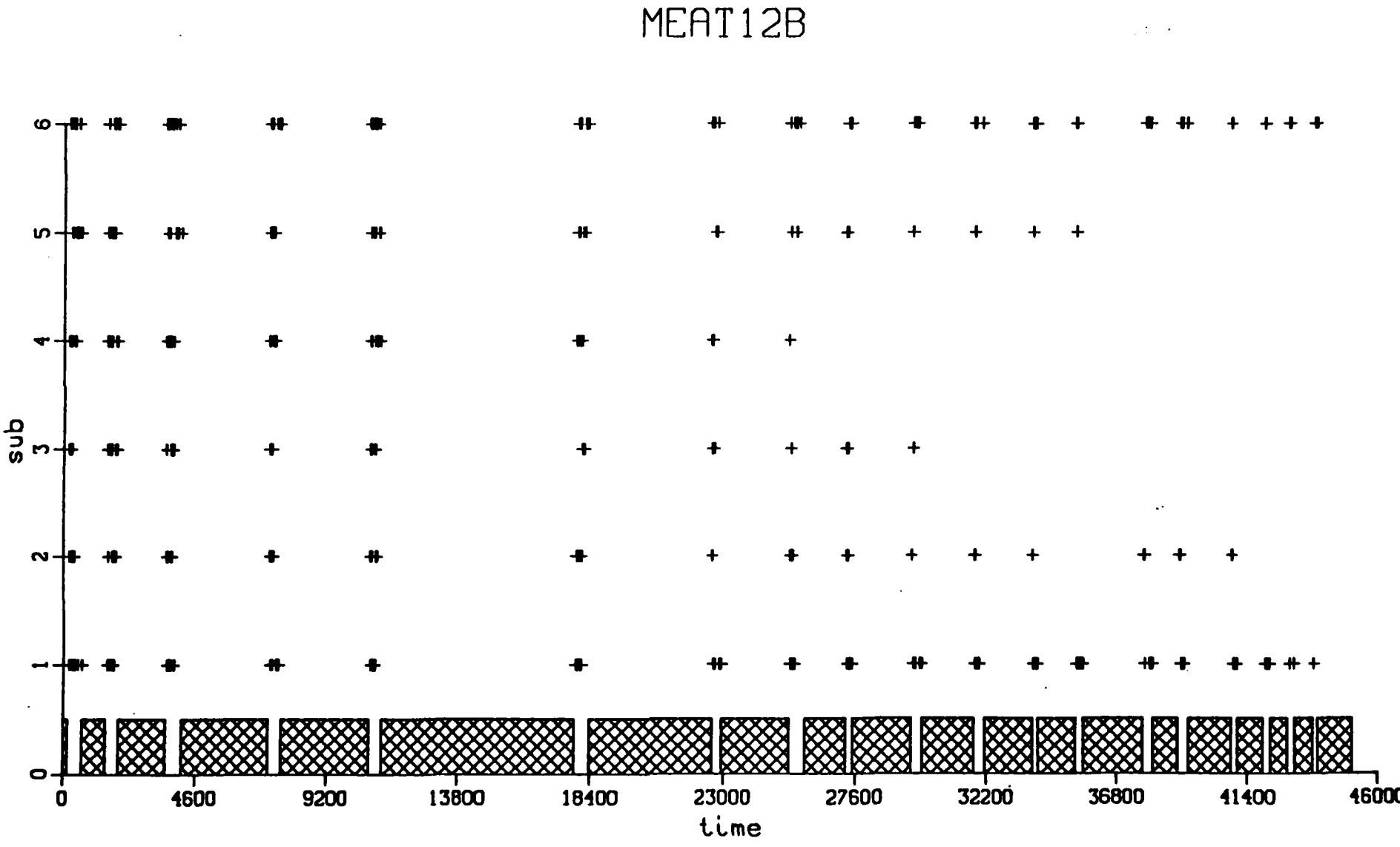

Figure 5 BIS Chart of Problem MEAT12B

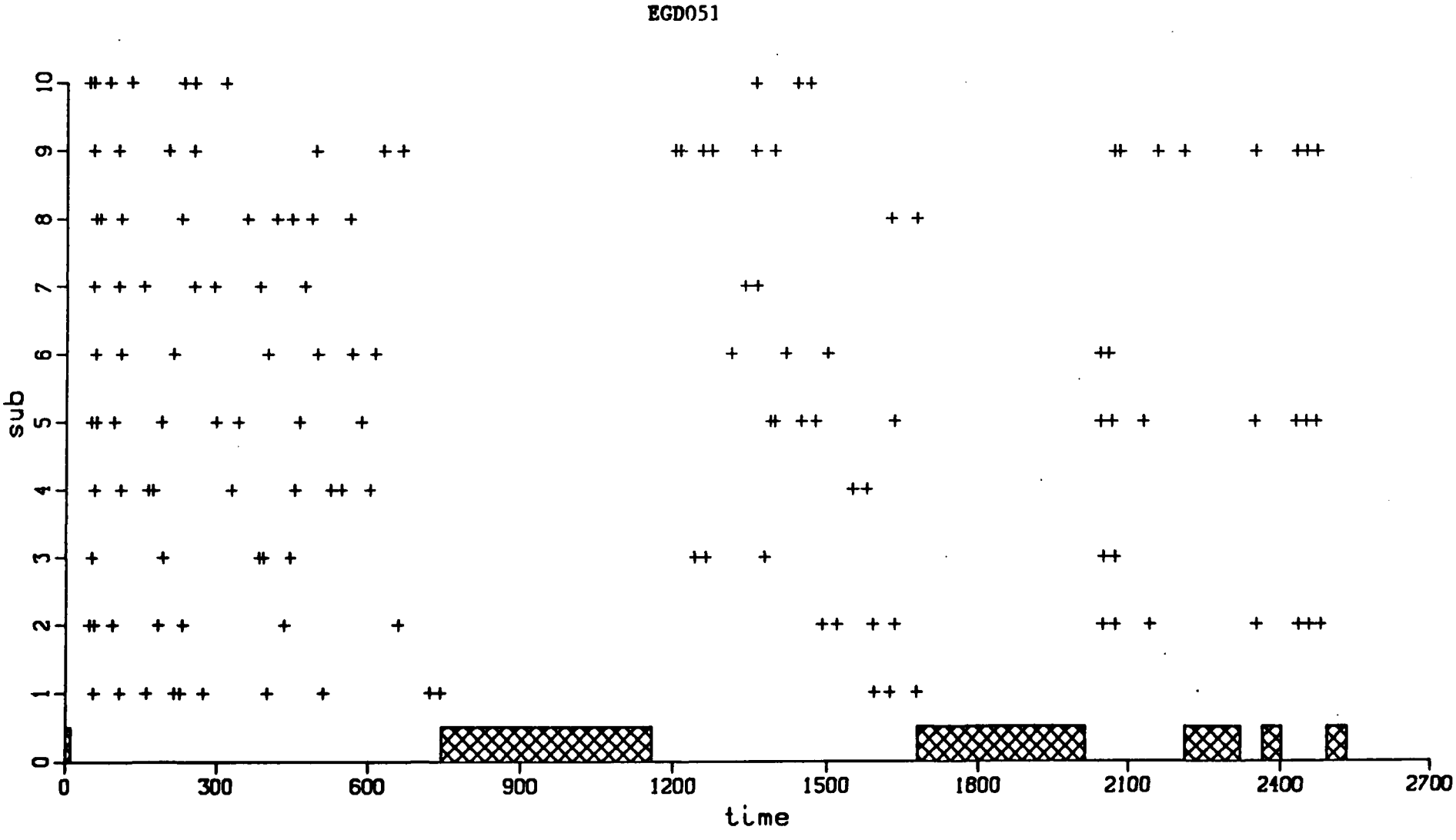

Figure 6 BIS Chart of Problem EGD051

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MSP-89-3

2. GOVT ACCESSION NO.:

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): Distributed Decomposition of Block-Angular Linear Programs on a Hypercube Computer

5. TYPE OF REPORT & PERIOD COVERED:

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): James K. Ho
S. Kingsley Gnanendran

8. CONTRACT OR GRANT NUMBER(s): N00014-89-J-1528

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
University of Tennessee, College of Bus. Admin.
Management Science Department
615 Stokely Management Center
Knoxville, TN 37996-0562

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS:
Office of Naval Research
Department of the Navy
800 N. Quincy Street
Arlington, VA 22217-5000

12. REPORT DATE: July 1989

13. NUMBER OF PAGES: 25 pages

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office):

15. SECURITY CLASS. (of this report):

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report):

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):

18. SUPPLEMENTARY NOTES:

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Algorithms based on the Dantzig-Wolfe decomposition principle for linear programs are implemented on an Intel iPSC-2 Hypercube computer with 64-processors. Computational results with block-angular linear programs from diverse applications are reported. They indicate that the approach of distributed computation on relatively inexpensive multiple processor computers may be very cost-effective for large, structured linear programs. It is also shown that by studying certain characteristics of the interaction among the master and subproblems, one can select algorithms that

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

best exploit the parallel processing environment.